محمدحسین عشوری

97149068

پروژه طراحی خزنده

شروع سریع:

scrapy crawl product -o result.json

با وارد کرد این کد در کامند لاین در مسیری که یک پروژه اسکرپی تعریف شده

و سه بار Enter زده تا مقادیر پیش فرض به برنامه داده شود و خزش صورت بگیرد.

در این پروژه لینک یکی از محصولات DIGIKALA را از طریق کامند به برنامه داده و برنامه به کمک فریم ورک Scrapy محصولات مشابه ای که در آن صفحه موجود است را بر میگرداند.

توضیح کد:

در متد init کلاس ProductSpider ما url ای که کاربر میخواهد برایش خزش کنیم را می گیریم (در صورت لینک نداشتن به صورت پیش فرض برنامه لینک پیش فرض را که برایش تعریف کردیم را خزش می کند)

```
class ProductSpider(scrapy.Spider):
    def __init__(self):
        url = input(
"""
-Enter URL of one of the Digikala's Product
OR
-Just press Enter to get sample Result
>>>""")
        if url != "":
            self.start_urls.clear()
            self.start_urls.append(url)
```

در این دو خط هم میبینم اگر ورودی ها ها را برای عمق سرچ و پهنای سرچ میگیریم

```
self.product_per_page = int (input("Enter number of product per page\n>>") or 16)
self.depth = int(input("Enter depth of crawling \n>>") or 10)
```

اگر ورودی مقدار داشت ما آن را لحاظ می کنیم ولی اگر نداشت برای مقدار پیش فرض مقدار ۱۶ را در نظر میگیرم به دلیل که در صفحه محصولات دیجی کالا حداکثر ۱۶ محصول مرتبط را نشان میدهد و ما حداکثر آن را در نظر میگیرم.

در خط بعدی هم عمق خزش را تعیین میکنیم که میخواهیم خزنده با تا چه عمق از محصول (لینک) مبدا را خزش کند ، که در این جا هم اگر کابر مقدار وارد کند ما آن را لحاظ می کنیم و اگر Enter خالی بزند ، مقدار پیش فرض ۱۰ را لحاظ می کنیم .

```
fa_num = {"۰": "0", "۱": "1", "۲": "2", "۳": "3", "۴": "4", "۵":
"5", "۶": "6", "۷": "7", "۸": "8", "۹": "9"}
name = "product"
url = "https://www.digikala.com"
start_urls = [
    "https://www.digikala.com/product/dkp-4149254",
]
```

در اینجا در خط اول یه دیکشنری ساخته ایم که key های آن مقدار یک عدد به فارسی و value آن مقدار همان عدد به زبان انگلیسی هست ، این برای این هست که وقتی قیمت یک محصول را وقتی بدست می آوریم آن هارا به اعداد انگلیسی تبدیل کنیم .

در خط بعدی هم نام را برای این ماژول انتخاب میکنیم این نام همان نامی است که ما در صفحه کامند لاین باید وارد کنیم که خزنده در پوشه اسپایدر رفته و فایلی با این نام را پیدا کند و اجرا کند.

خط بعدی هم url بعدی هست برای اینکه برای لینک های دیگر محصولاتی که میخواهیم خزش کنیم نیاز داریم به این دلیل که در صفحات html لینک های داخل صفحه که به خود وب سایت لینک داده می شوند قسمت اولشان حذف می شود که ما باید به یک طریقی این url اصلی را به اول لینک اضافه کنیم تا دیگر محصولات صفحه را خزش کنیم

و آدرس پیش فرش را هم در همان آرایه start_urls میگذاریم .

```
def convert_num(self, num_str):
    ans = ""
    for char in num_str:
        temp = self.fa_num.get(char)
        if temp != None:
            ans += temp
        else:
            ans += char
    return ans
```

این متد هم درواقع مقدار های عددی فارسی را با کمک دیکشنری که در بالا ذکر کردم به معادل انگلیسی آن بتدیل میکند.

در متد parse ما یک متغیر response داریم که باید تمام اطلاعات صفحه را از بیرون بکشیم

در خط اول عمق حال حاضر را بدست می آوریم ، با این کار میتوانیم از خزش محصولاتی که از عمق مورد نظر ما پایین تر هستند جلو گیری کنیم و فقط اطلاعات آن محصول را بدست آوریم و دیگر فرزندانش (محصولات مرتبط) را خزش نکنیم

```
def parse(self, response):
    depth = response.meta.get('depth')
    category =
response.xpath('//*[@id="content"]/div[1]/div/article/section[1]
/div[1]/div/div/div/a[2]/text()')
    name_en =
response.xpath('//*[@id="content"]/div[1]/div/article/section[1]
/div[2]/div[2]/span/text()')
```

در خط بعد متغیر category و en_name را داریم که نام انگلیسی محصول را ذخیره می کند.

به این دلیل که این متغیر ها در بعضی از محصولات موجود نیستند ما نیاز داریم که با یک سری شرط از صحت اطلاعات اطمینان پیدا کنیم.

```
yield {
        "name":
response.xpath('//*[@id="content"]/div[1]/div/article/section[1]
/div[1]/div/h1/text()').get().strip(),
        "en-name": name_en.get().strip() if len(name_en) else
"",
        "price":
self.convert_num(response.xpath('//*[@id="content"]/div[1]/div/a
rticle/section[1]/div[2]/div[3]/div/div[1]/div[1]/div[11]/div[2]
/div/text()').get().strip()),
        "category": category.get().strip() if len(category) else
"",
        "url": response.url,

    }
```

در تکه کد بالا ما به وسیله yield یک دیکشنری از اطلاعاتی که مورد نیاز مان بود از تابع بیرون می اندازیم در بیرون از تابع این مقدار گرفته شده هم چاپ میشوند و هم این مقدار با فرمت مورد نظر ما در انتها در یک فایلی چاپ خواهند شد.

اطلاعاتی که در این دیکشنری هستند هم همگی به جز en-name و category برای همه وجود دارند به همین علت برای آنها از شرطی برای صحت اطلاعات ایجاد نکردیم.

مسیر های xpath موجو هم به راحتی میتوان در حالت inspect مرورگر مشاهده نمود و مسیر نسبی را در آن نوشت.

بدست آوردن لینک های عمق بعدی

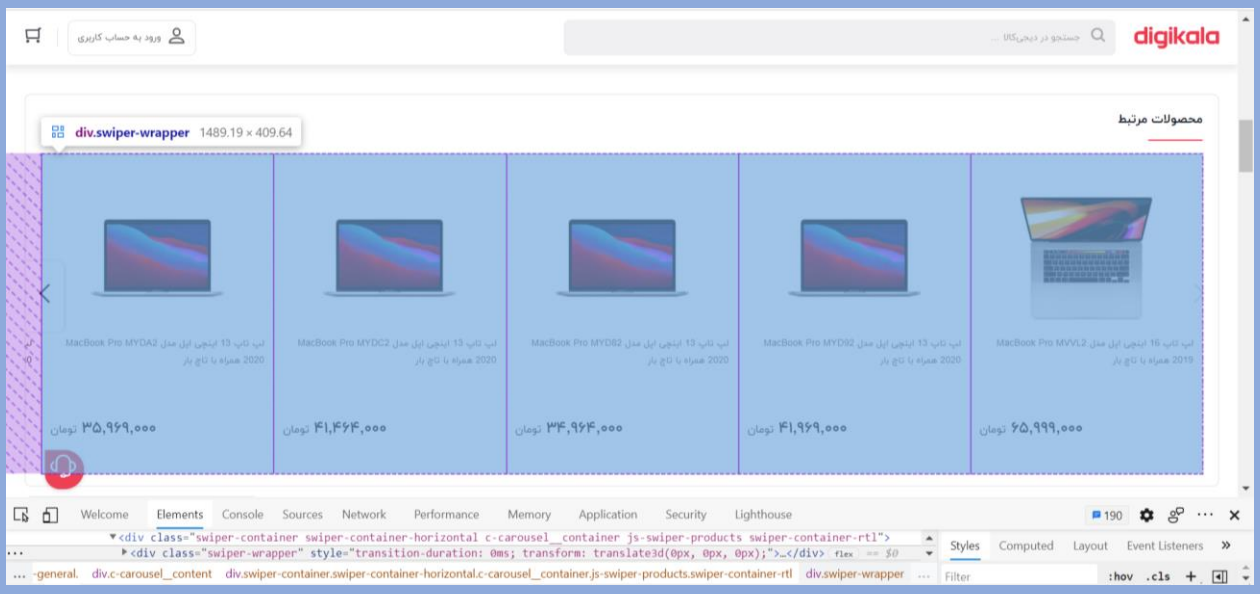
```
if depth < self.depth:
    product_links =
response.xpath('//*[@id="content"]/div[1]/div/div[3]/div[2]/div/
div/div//a')
    links = []
    product_links = product_links if len(product_links) <
self.product_per_page else product_links[

:self.product_per_page]
    for item in product_links:
        links.append(self.url +
"/".join(item.attrib["href"].split("/")[:3]))

    yield from response.follow_all(links, callback=self.parse)
```

در این قسمت اول عمق حال حاضر چک میشود که از عمق مورد نظر ما بیشتر نشده باشد.

در این قسمت از تابع parse ما بدنبال لینک های عمق بعد هستیم .



برای این کار به این قسمت از یکی از محصولات DIGIKALA رفته و در قسمت محصولات مربط این مسیر مربوط به این ناحیه را پیدا میکنیم و آن را ذخیره می کنیم

مسیر مورد نظر :

```
//*[@id="content"]/div[1]/div/div[3]/div[2]/div/div/div
```

تا این قسمت ما به تمامی ابجکت کلی محصولات دسترسی خواهیم داشت

ولی چون ما تگ a از این محصولات را می خواهیم (چون لینک آن محصول در آن تگ می باشد) به همین دلیل به آخر آن

```
//a
```

اضافه می کنیم که نتیجه به این صورت خواهد شد :

```
response.xpath('//*[@id="content"]/div[1]/div/div[3]/div[2]/div/
div/div//a')
```

با این روش ما به همه لینک های آن مجموعه دسترسی داریم

```
for item in product_links:
    links.append(self.url +
"/".join(item.attrib["href"].split("/")[:3]))
```

حال با این تکه کد url هایی را که بدست آوردیم به قسمت اولشان که digikala.com هست متصل میکنیم که در خزش های بعدی url کامل باشد.

در آخر هم لینک های بدست آمده را از تابع بیرون می اندازیم و به صورت بازگشتی تمامی لینک های موجود تا عمق مشخص شده خزش می شوند.

توجه :

- در حین عملیات به دلیل مشکلات شبکه خود یا به دلیل محدودیت هایی که خود سایت ها میگذارند ممکن است برخی از درخواست پاسخ داده نشود و برخی از لینک ها محاسبه نشوند ، از طرفی برخی از محصولاتی که ما در ابتدا خزش میکنیم ممکن است در لیست محصولات مرتبط دیگر محصولات ظاهر شوند واگر جلو لینک های تکراری را نگیریم خطای سرریز یا stack over flow پیش می آید ، که خوش بختانه با توجه به ویژگی هایی که در خود فریم ورک scrapy موجود می باشد خودش قبل از صدا زدن متد parse لینک مورد نظر را چک میکند که آیا این لینک قبلا برسی شده است یا خیر و از آن جایی که لینک محصولات unique هستند این موضوع صرفا بر اساس لینک قابل انجام هستند.
- در برخی از سیستم ها در هنگام ذخیره کردن اطلاعات در به فرم json به این دلیل که از کارکتر های فارسی پشتیبانی نمیشود فرمت آنها تبدیل میشوند که اگر آن فایل با یک editor مناسب باز شود اطلاعات فارسی هم قابل رویت خواهد بود